FORTH USER

(formerly ACE USER)

vol. 2 no.2/3 **1984**

Contents:

EDITORIAL - John Noyce

Well its been a long time in preparation, but I hope you find it worthwhile. Part of the delay was due to moving house, others - well, its been a nice summer...

This issue is, with the exception of Garry Knight's Spectrum Forth comparison, entirely on the Jupiter Ace. Jupiter Cantab may have crashed in 1983, but the Ace lives on, indeed there has been a renewal of enthusiasm for it this year, as Boldfield have sold off bankrupt stocks, and assembled an impressive list of add-ons. More recently Boldfield have taken the plunge and produced some new software for the Ace - anyone want to review these items? Just in passing, we still have some copies of the Users Club software left (in some cases very few).

Next year's issues will still, I suspect, be mainly on the Ace, but hopefully some more articles on other machines using Forth will be forthcoming. All articles, letters, etc, always welcome.

Finally a note on reproduction quality - whilst some of the articles were retyped, I have reprinted 'as is' as this saves retyping and removes possibility of error in the retyping of listings. If contributors could use a typewriter with a decent ribbon that would be helpful - or at very least use a black biro if writing by hand.

See you next year

Ps. Subns are due - still £7 for 3 issues.

FORTH USER is edited and published by

JOHN L.NOYCE, Publisher, PO Box 450, Brighton BN1 8GR, U.K.

Subscription: £7 pa (overseas:£8)

There are some six or seven Spectrum FORTHs currently on the market - Abersoft FORTH, Artic FORTH, Thurnall Engineering FORTH Compiler - the three reviewed here - two by Mike Hampson (integer and floating-point), and one or two others whose names escape me at the moment. Of the first three, Abersoft and Artic are a mixture of FIG and FORTH-79, while the third is very unusual indeed. For this reason I am going to do a comparison of the first two in terms of the commands supported and the differences between them, and then I'll explain just why TE FORTH Compiler is so strange.

The first two programs arrive in a library case with cassette and manual(s). Loading is by LOAD "" CODE for Artic and LOAD "" for Abersoft. Loading time is only 1 to 1½ minutes and both auto-start with a copyright line. At this point Artic also tells you that you have 23479 bytes free. You can find out how much memory you have available at any time by entering MEM. The equivalent command with Abersoft is FREE, but you then have to use the FORTH 'print' command (.) - which, with no user-defined words in the dictionary returns 18919 bytes. The manuals are very comprehensive, Abersoft at 28 pages, Artic at 44 pages but with a separate Editor manual. Both editors are standard FORTH line editors, but whereas Abersoft's is included in the dictionary, you must load Artic's from tape. I'll explain how you do this later.

Programs are written to and edited on FORTH screens of 16 lines by 64 characters, but Artic only has one of these resident in memory, whereas Abersoft has an internal 11-screen ram-disc. Screens may be selected for editing by n LIST or n CLEAR, but with Artic, if the screen you want to edit is not the one currently in memory, it will try and load it from tape. In addition to the line editor, Artic also has a separate screen editor, which works like this - if, in immediate mode, you type in a command and execute it, provided it does not clear the screen you can move the cursor up to the start of the command with the normal Caps-shifted cursor keys, then press shift-1

to edit, hold it down, and the command is copied down to where the cursor originally was. Pressing ENTER will then execute the command a second time. I think this is one of Artic's best features, because, if you define a word incorrectly and get an error message, you can cursor-up, copy the bits you want to keep, typing in the correction as you go.

As far as tape-handling is concerned, both programs have their good and bad points. With Abersoft you can save an extended version of the dictionary, including your own defined words, and you can save, load and verify the whole 11K ram-disc. However, you can only save the whole 11K, which takes time. It is fairly easy, if you speak Z80 assembler, and you have a copy of the ROM disassembly, to define your own single-screen load and save routines. With Artic's save, you can only save one screen at a time (the one in memory), which means that you have to plan your programs very carefully if you intend to use more than one screen-full of definitions. To save you use the word FLUSH and to load you use n LIST, where n is the number of the screen to be loaded, so you need to keep a written record of the numbers of the screens you have previously saved. A cassette recorder with a tape-counter is a help, too. Also you need to make sure that the screen you are trying to load does not have the same number as the screen in memory, or LIST won't even look at the tape, but will list the one in memory to the screen. Also there is no routine to verify, unless you write a machine-code version yourself.

Both Abersoft and Artic support most of the Spectrum's graphic commands, CIRCLE being the only one that Abersoft lacks, while Artic doesn't have SCREEN$, ATTR or POINT. Because the FORTH word OVER works differently from the Spectrum graphic command, both versions use GOVER, and I regularly forget this and use OVER and wonder why it doesn't work. Character definition with Artic is as easy

as stacking eight numbers, then the character code, then using the DEF command. With Abersoft you need to use the FORTH variable UDG to work out where to c! your numbers.

The BEEP command is also supported by Artic, but I had difficulty getting it to work. Abersoft has the word BLEEP which works differently from the Spectrum BEEP, being a self-contained machine-code routine. You can create some amazing arcade-style sounds using this word in loops. On the whole, Abersoft's screen and sound commands are more useful for writing games than Artic's and the dictionary-save ability makes it easy to run them too. Both versions also have CLS and AT, and Artic also has HOME, which puts the cursor at Ø,Ø without clearing the screen.

Another interesting difference between the two packages is the use of the BREAK key. Abersoft uses two of them, the normal BREAK and Caps-shift-1. The difference is that when using the regular Caps-Space key during printer or cassette operations, you are usually dumped into BASIC, whereas shift-1 leaves you in FORTH. The only problem is that, while both keys stop the VLIST command, neither of them will BREAK into a program while it is running, so you need to be careful about executing definitions that end in Ø UNTIL. You can, however, check for caps-shift-1 by including the words ?TERMINAL IF QUIT THEN in your loops. The Artic break key is the standard one and BREAKs programs as well as VLISTs.

Also, while on the subject of keys, both versions have KEY which waits for a key-press and leaves the ASCII code on the stack; Abersoft also has INKEY which doesn't wait, but returns Ø if no key is being pressed. The printer commands also differ in that Artic uses 1 PRINT ! to divert all printed output to the printer, Ø PRINT ! to re-divert it to the screen, and COPY to do a screen-dump. Abersoft uses the variable LINK which echoes all screen output to the printer if it contains a non-zero value.

Abersoft also considers the machine-code programmer by making available the words PUSHDE, PUSHHL, POPDE and POPHL, and by listing in the manual which registers the FORTH uses internally, and which must be preserved across word definitions. The only way I could find to implement my own m/c routines is to CREATE a header, manually assemble bytes using c! and then change the code-field pointer to point to the first byte of the code. The word CALL could have usefully been implemented to avoid this.

I'll quickly run through the other differences between the two FORTHs. Both use numbered error codes which are listed in the manuals; both support double-precision mathematics. Abersoft also has a complete CASE structure and a word SIZE which places on the stack the size of the dictionary defined so far. If you mis-define a word, Artic will let you FORGET a partially-completed definition, but Abersoft requires SMUDGE before FORGET in this case. Both will allow you to exit from FORTH - Artic with BYE which resets the entire system (effectively RAND USR Ø) and Abersoft with MON which returns you to BASIC with the error message STOP at line 2. You can also restart Abersoft with RAND USR 24132 for a warm start (with user-defined words intact), or RAND USR 24128 for a cold start.

I've used both packages for some time (my first encounter with FORTH was Artic's FORTH for the ZX81, which is almost exactly the same as Spectrum FORTH), and coming to any decision as to which is the better is difficult - it rather depends on what you want. Both support enough commands to run most applications, with Abersoft having better graphics and sound handling. If you can program machine-code either will do, and the prices are similar enough to make little difference to most pockets. Sinclair support Artic's version, but I think that Abersoft has the edge.

Now, on to the last package - Thurnall Engineering Forth Compiler. I've deliberately left it until last as it is nothing like the other two. The internal structure may be FORTH-like, but the user-interface certainly isn't. When you load the program you are presented with a menu, with options to create, edit or delete a command, load or save a program to tape, compile a program, delete the

compiler, or list the directory (sic!) of commands compiled so far. Everything you want to do has to be done through this menu system and no command can be executed in immediate mode, except after compiling it, and even then it sometimes doesn't seem to work. The only advantage to this version of FORTH is that, once you have written your program you can dump the compiler out of memory, leaving a block of compiled code at the top of RAM which you can access with a single USR call.I must admit, I only tried this once, but it didn't work for me.

The program comes in an impressive looking library case and takes about 1½ minutes to load. The manual runs to 32 pages, half of which is a list of the 63 commands available to the user. Also included is an adhesive overlay which you stick above the number keys. The top-row keys are used by the program as 'function' keys, 8 and 9 performing cursor-left and right, Ø for delete, and the others are used in creating commands. On choosing the menu option to create a new command (by pressing 'c') you are asked for a name, which must be at least two letters long. If the first word in your new definition is to be BEGIN you must press key '2' (marked ! and CALL on the overlay), then type BEGIN. Each command in a definition must be CALLed in this way and numbers must be preceded with ENTER (key 3 on the overlay).

I don't want to spend too much time explaining how to create new commands - instead I'll give you a sample listing from the manual:-

```
!START !BEGIN !INKEY !TEST !WHILE !OVER !OVER
!PLOT !REPEAT
```

The exclamation mark in the listing is how the CALL function is shown on the overlay.

The 63 available commands take up less than half of the screen when listed, and list alphabetically, with user definitions shown underneath as 'programs'. There are very few graphic commands, and almost all of the names

used are non-standard. For instance, the word FETCH is used in place of @. There is an extension dictionary on the tape, which comprises 6 words - AND, DEPTH, MAX, MIN, OR and XOR - four of which I would consider to be absolutely necessary for writing anything useful.

Because the dictionary is so small and non-standard, and the program does not support many Spectrum BASCIC commands, and, on top of this, the price - I can't recommend it and, further, I can't even see why Thurnall Engineering thought anyone might want to buy it.
One problem I had was that I had to use a sharp implement to force the cassette out of the library case. Having reviewed the program I have put the cassette back in the case, and, as far as I'm concerned, it can now stay stuck there!

Garry Knight
3ØA Stanton House
Thames Street
London, SE1Ø 9DJ

6.6.84

SIMPLE ROBOT

Here is the design of a simple BBC-type buggy I built for my Ace. It uses either the Essex Micro Electronics sound board 1/0 port, or the spare 1/0 port of the sound board in the last issue of Ace User.

Components	Qty
BC107 transistor	5
1K resistor	5
IN4004 Diode	5
SPCO 6V Relay	5
3V motor	2
wheels for above	2

Edge connector, connecting wire for buggy (4-6 core).

The motors, wheels and relays all came from Greenweld Electronics. The motors cost £5.95 and came already geared and cased.

The electronic circuit is not complicated. Make sure you connect the relays to the 1/0 port in the right order, or the motor's power supply may be short circuited.

If the motor's power supply (I used 2x HPII's) gets mixed up, the robot will go backwards when you tell it to go forwards etc.

The actual robot can be made as basic or as advanced as you wish- mine was just a Meccano frame enclosing the motors.

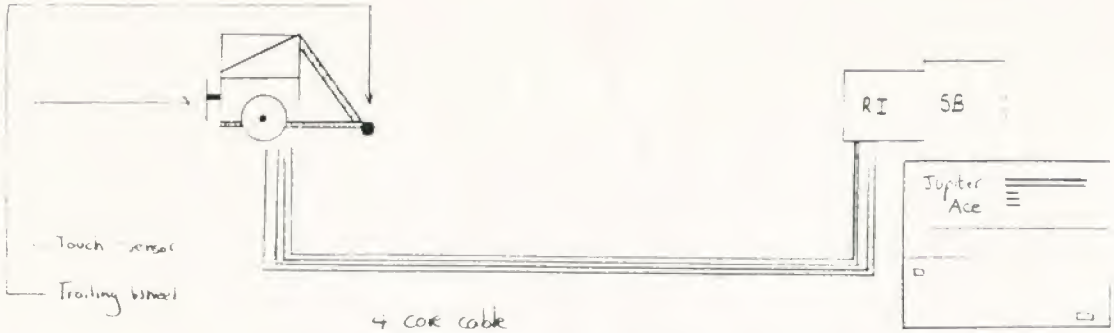
To run the software, the correct version of CONTROI should be used. Before the robot can be controlled, the word INIT must be used. The relays can be switched on and off and the buggy steered by using a number from the direction table and the word SWITCH.

RUN enables you to guide the robot using the cursor keys, while RECORD will remember up to 200 steps, beginning when you press a key. PLAY will repeat the sequence.

The buggy can be greatly enhanced: I give the touch sensory as an example:

Direction Table	
STOP	Ø
FORWARD	31
BACKWARDS	1
LEFT	25
RIGHT	7

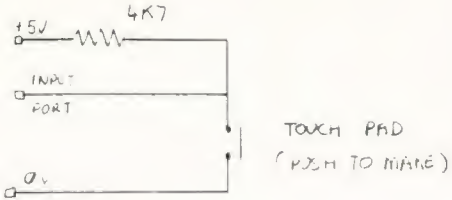
JOHN KENNEDY.



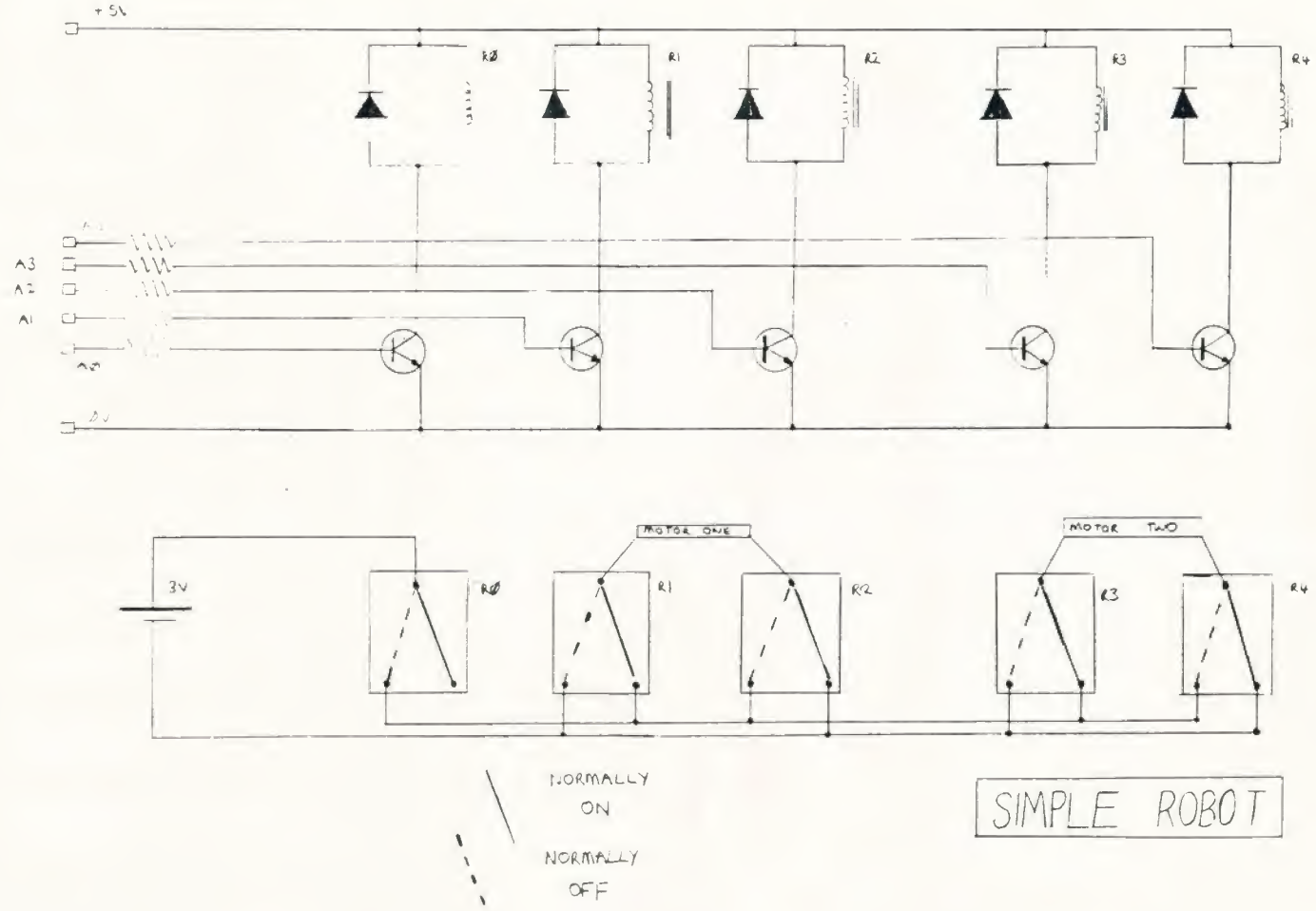
RI Robot Interface + 3v Supply

SB Sound Board

EXAMPLE TOUCH SENSOR



SIMPLE ROBOT


+ 5V
R0
R1
R2
R3
R4
A3
A2
A1
3V
MOTOR ONE
MOTOR TWO
NORMALLY ON
NORMALLY OFF
SIMPLE ROBOT

```
: CONTROL1
( ESSEX MICRO ELECTRONICS)
( VERSION)
SWAP 253 OUT 255 OUT
;

: CONTROL2
( LAST ISSUE SOUND PORT)
( VERSION)
SWAP 221 OUT 223 OUT
;

: SWITCH
( ACTIVATE RELAY)
14 SWAP CONTROL1 or 2
;

: INIT
( PREPARE RELAYS)
7 64 CONTROL1 CLS
                 or 2
;

CREATE STORE 200 ALLOT

0 VARIABLE POS

0 VARIABLE FLAG
: STOP
0 SWITCH ." stop      "
;

: F
." forward"
31 SWITCH
;

: B
." back      "
1 SWITCH
;

: R
." right     "
7 SWITCH
;

: L
." left      "
25 SWITCH
;

: INC
CR POS @ 1+ POS
!

: END
0 SWITCH 5 0
DO
 100 100 BEEP 500 100
 BEEP
LOOP
```

```
: RUN
CLS ." USE CURSOR KEYS, (0=STOP
)      PRESS '1' TO STOP PROGRA
M"
BEGIN
 10 10 AT INKEY DUP
 48 - DUP DUP DUP
 DUP 0 =
 IF
  0 SWITCH ." stop      "
 THEN
 5 =
 IF
  L
 THEN
 6 =
 IF
  F
 THEN
 7 =
 IF
  B
 THEN
 8 =
 IF
  R
 THEN
 0= 0=
 IF
  9 SPACES
 THEN
 INKEY 49 =
UNTIL
5 0
DO
 100 100 BEEP
LOOP
END
;

 RECORD
54 FLAG ! CLS ."          RECORDING
PATTERN"
CR CR CR 0 POS
!
BEGIN
 INKEY DUP 0=
 IF
  DROP FLAG @
 ELSE
  DUP FLAG !
 THEN
 DUP DUP DUP DUP DUP
 48 =
 IF
  STOP 0 STORE POS @
  + C! INC
 THEN
 53 =
 IF
  L 5 STORE POS @
  + C! INC
 THEN
 54 =
 IF
  F 6 STORE POS @
  + C! INC
 THEN
 55 =
 IF
  B 7 STORE POS @
  + C! INC
 THEN
 56 =
 IF
  R 8 STORE POS @
  + C! INC
 THEN
 49 =
 IF
  42 STORE POS @ +
  C! INC
 THEN
 1000 200 BEEP POS @
 200 = STORE POS @
 1- + C@ 42 =
 OR
UNTIL
." FINISHED!!!"
END
;

: PLAY
 CLS ." PLAYING BACK RECORDED P
ATTERN     PRESS -0- TO START PLA
SE."
 BEGIN
  INKEY 48 =
 UNTIL
 100 100 BEEP 0 POS
 !
 BEGIN
  10 10 AT STORE POS
  @ + C@ DUP DUP
  DUP DUP 0=
  IF
   STOP
  THEN
  5 =
  IF
   L
  THEN
  6 =
  IF
   F
  THEN
  7 =
  IF
   B
  THEN
  8 =
  IF
   R
  THEN
  1000 200 BEEP INC POS
  @ 200 = STORE POS
  @ + C@ 42 =
  OR
 UNTIL
 CLS 10 10 AT ." FINISHED!"
 END
;
```

```
( SOME MORE CONTROL STRUCTURES )

: ERROR                                 ( ERROR_NO -- )
  15421 ! ABORT
;

2 COMPILER STARTLOOP                    ( U_INDEX -- )
  HERE 2 ALLOT 6                        ( usage; STARTLOOP ... ENDLOOP )
RUNS>                                   ( performs words between S and E )
  SWAP ?DUP                             ( u_index times )
  IF                                    (  )
    1- R> SWAP >R >R                    ( NOTE; Ø STARTLOOP will )
    DROP                                ( iterate zero times! )
  ELSE
    @ R> + >R
  THEN
;

2 COMPILER ENDLOOP                      ( -- )
  6 -                                   ( see STARTLOOP )
  IF                                    (  )
    HERE OVER - OVER !                  ( I will return no. of iterations )
    HERE - ,                            ( left to perform within loop )
  ELSE                                  ( hence same restrictions with )
    5 ERROR                             ( R> and >R and EXIT as DO ... )
  THEN                                  ( LOOP )
RUNS>                                   (  )
  I'                                    ( must of course be correctly )
  IF                                    ( nested )
    R> R> 1- >R SWAP
    @ + >R
  ELSE
    DROP R> R> DROP >R
  THEN
;

: EXITLOOP                              ( -- )
  R> R> DROP Ø >R >R                    ( terminates loop at next )
;                                       ( ENDLOOP cf. LEAVE )

2 COMPILER FOREVER                      ( -- )
  1 -                                   ( directly replaces Ø UNTIL )
  IF
    HERE - 2- ,
  ELSE
    5 ERROR
  THEN
RUNS>
  @ R> + >R
;                                       ( Gordon R. Charlton )
```

AN ALTERNATIVE CASE

P.Goff

This version forces a run-time branch to the procedure selected by the number on top of the stack;the program continues through the word ENDCASE.

A string of '(NAME).(N) ENDCASE' is written into the buffer line, starting at 9953 (it is ESSENTIAL to leave the first buffer byte clear).

This approach has the advantage of readibility and ease of editing. Many implementations are possible;the example below is a simple version dealing with up to 9 cases.

Note that at LINE in CASE the program flow jumps to ENDCASE;any words after are ignored as the program effectively ends if flow returns to CASE.

```
1.  CREATE (NAME) eg CONTROL
(maximum number of branches) C,
(number of characters in (NAME) + 10 C, eg 17 for CONTROL
(NAME character by character) C,
    eg 67 79 78 84 82 79 76  for CONTROL
46 C, 48 C, for CONTROL.0
32 C, 69 C, 78 C, 68 C, 67 C, 65 C, 83 C, 69 C,
       for  CONTROL.0 ENDCASE

2.  Define 9953 CONSTANT BUF

3.  Define a word to change the stack number to 0 or the maximum
branch number,if negative or greater than the maximum eg:

      : CHECKCASE
        DUP 0<
            IF
             DROP 0
            THEN
        DUP 3 PICK C@ >
            IF
             DROP DUP C@
            THEN
      ;

4.  Define a word to put the stack number into the (NAME) string eg:

      : SETCASE
        48 + OVER 1+ DUP C@ 8 - + C!
      ;

5.  Define a word to load a string into the buffer,at 9953 eg:

      : FORCE
        2 + DUP 1 - C@ 0
            DO
             DUP I + C@
             BUF I + C!
            LOOP
        DROP
      ;
```

6. *Define a word to control branching eg:*

```
: CASE
  0 SETCASE SWAP CHECKCASE
  SETCASE FORCE LINE
;
    ( the first SETCASE defaults to CONTROL.0 , the second
      takes N of the stack to give  CONTROL.N )
```

7. *Define a word ENDCASE , linking to the rest of the program eg:*

```
: ENDCASE
  ..... as appropriate.....
;
```

---------- *REFINEMENTS AND EXTENSIONS*

The number of cases need not be limited to 9; the string could contain a pair of ASCII codes for numbers up to 99. A possible approach would use an expression such as:-

```
: 2D
  10 /MOD 48 +
  SWAP 48 +
;
```

to convert the stack number into 2 string digits.

A refinement would be to set the string 'ENDCASE' in its own carrier word, then use it with a variety of case (NAME)s. Routines to write it in the buffer after (NAME.N) would be required.

Words to construct the strings from keyboard entry could be written.

---------- *USE OF THIS PROCEDURE*

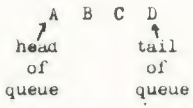
To use this approach, the program is essentially written in segments, eg:

```
: ------ ( number on stack ) (NAME eg CONTROL ) CASE

: ENDCASE ----------    ;
```

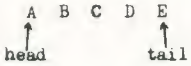
QUEUES

In the same way that a stack is a last-in first-out (LIFO) data structure, a queue may be described as a first-in first-out (FIFO) structure. So the first item added to a queue is the first available from the queue, and the most recently added is the last availabe.

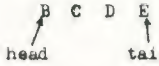
Given a queue which contains four elements;



adding an item E to the queue would leave it as;



and subsequently removing an item would give;



The following words add the data type QUEUE to the Ace vocabulary, and also the necessary queue manipulation commands.

QUEUE

use; n QUEUE queuename

Declares a queue called queuename, which will hold at most n elements (16 bit numbers: notes are given in the listing for conversion to 8 bit elements).

QLENGTH

Given the address of a queue variable on the stack, will return the maximum number of elements it can hold. i.e.;

```
10 QUEUE A
A QLENGTH .
```

would print; 10 OK

QSIZE

Given the address of a queue, will return the number of items currently in the queue.

QEMPTY

Given the addr. of a queue will return True (1) if the queue has no elements, and False (Ø) otherwise.

QFULL

This returns True if the specified queue can accept no more items, and False otherwise.

Q@

This will return the next available item on the specified queue, and remove it from the queue.

Note -- attempting to use this on an empty queue will cause ERROR 15 - queue underflow.

Q!

Adds a data item to the named queue i.e.;

```
123  A  Q!
```

will add the number 123 to the queue called A .

Note -- attempting to add an element to a queue which is full causes ERROR 16 - queue overflow.

QCLEAR

Empties the specified queue, so that it contains zero elements.

By way of an example of queue manipulation I also include;

QNEXT

Fetches the next available element of the queue using Q@ and adds it back onto the tail of the queue using Q! , whilst leaving a copy on the stack.

This is used by;

.Q

which will print the contents of the named queue non-destructively, in same way as .S displays the stack. i.e.;

```
10  QUEUE  A
5  A  Q!
6  A  Q!
7  A  Q!
A  .Q
```

will print; 5 6 7 OK , and;

```
A  QCLEAR
A  .Q
```

will print; queue empty OK .

This occupies less than ½k (497 bytes) altogether*, and thus will fit into an unexpanded Ace with plenty of room to experiment.

Gordon R. Charlton

*Without comments.

(QUEUE STRUCTURE WORDS)

```
DECIMAL

: ERROR                          ( ERROR_NO -- )
  15421 ! ABORT
;

DEFINER QUEUE                    ( MAX_Q_SIZE -- )
  1+ DUP , Ø ,                   ( defined word returns Q_ADDR )
  Ø , 2 * ALLOT                  ( for 8-bit omit 2 * )
DOES
;

: HEAD                           ( Q_ADDR -- OFFSET_TO_HEAD_OF_Q )
  2+ @                           ( 8-bit use C@ instead of @ )
;

: TAIL                           ( Q_ADDR -- OFFSET_TO_TAIL )
  4 + @                          ( 8-bit use C@ instead of @ )
;

: ITEM                           ( Q_ADDR , OFFSET -- ADDR_OF_ITEM )
  2 * 6 + +                      ( for 8-bit omit 2 * )
;

: WRAP                           ( OFFSET , Q_ADDR -- WRAPPED_OFFSET)
  @ DUP ROT + SWAP               ( modifies offset to fit the size of)
  MOD                            ( the queue )
;

: QSIZE                          ( Q_ADDR -- MAX_Q_SIZE )
  @ 1-
;

: QLENGTH                        ( Q_ADDR -- NO_OF_ITEMS_IN_Q )
  DUP TAIL OVER HEAD -
  SWAP WRAP
;

: QEMPTY                         ( Q_ADDR -- ISEMPTY )
  QLENGTH Ø=
;

: QFULL                          ( QADDR -- ISFULL )
  DUP QSIZE SWAP QLENGTH -
;

: Q@                             ( Q_ADDR -- Q_ELEMENT )
  DUP QEMPTY
  IF
    15 ERROR
  ELSE
    DUP HEAD OVER OVER ITEM
    @ ROT ROT 1+ OVER            ( 8-bit use C@ instead of @ )
    WRAP SWAP 2+ !
  THEN
;
```

```
: Q!                              ( DATA , Q_ADDR -- )
  DUP QFULL
  IF
    16 ERROR
  ELSE
    SWAP  OVER  DUP  TAIL  ITEM
    !  DUP  TAIL  1+  OVER        ( 8-bit use C! instead of ! )
    WRAP  SWAP  4  +  !
  THEN
;

: QNEXT                           ( Q_ADDR -- NEXT_ELEMENT )
  DUP  Q@  DUP  ROT  Q!
;

: .Q                              ( Q_ADDR -- )
  DUP  QEMPTY
  IF
    ." queue empty"
  ELSE
    DUP  QLENGTH  0
    DO
      DUP  QNEXT  .
    LOOP
  THEN
  DROP
;
```

- - - -

JUPITER ACE STACK MANIPULATION

B. Thornton

While programming my ACE it became apparent to me that it lacks any words which push numbers down the stack. It is always possible to write words on an ad hoc basis but I enclose three examples that I have found useful. They allow "rolling" and picking in reverse i.e. down the stack and switching of numbers in the stack. Full control over the stack enables you to use variables and constants less by storing numbers out of the way, but still on the stack, until required. I offered these to "Your Computer" months ago, without success, so if you think they are worth printing you are welcome to do so in the Users' newsletter.

→

B. THORNTON — JUPITER ACE STACK MANIPULATION.

1 Reverse rolling

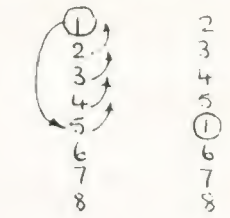
```
: RROLL
    DUP 1 DO
    DUP 1+ ROLL
    SWAP LOOP
    DROP
;
```

Examples

STACK

Before — After

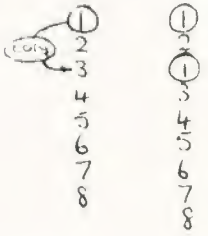
(5 RROLL)



2 Reverse picking

```
: RPICK
    SWAP DUP
    ROT RROLL
;
```

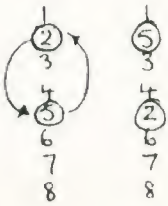
(3 RPICK)



3 Switching

```
: SWITCH
    DUP ROT DUP
    3 + ROLL ROT
    1+ RRROLL SWAP
    1+ ROLL SWAP
    RROLL
;
```

(2 5 SWITCH)



Dear Mr Noyce,

I enclose a little routine which other ACE users may find useful. The idea is that the word LREF browses through the CONTEXT voabulary finding references to the word you are interested in:- i.e. LREF PPUT lists all words containing a reference to PPUT. LREF is very useful with a large program, where locating references by any other means is a real bind.

Glad that you are carrying on; keep up the good work.

Yours sincerely,

H.G. Woods

LREF - Find references in Context Vocab

```
: WDNM
-1 DUP 4 - SWAP DUP
C@ - 4 -
DO
  I C@ 127 AND EMIT
LOOP
SPACE
;

: LREF
FIND ?DUP
IF
  ." = "
  CONTEXT @ 3 - CONTEXT
  @
  BEGIN
    @ OVER OVER = 0 =
  WHILE
    DUP 4 - DUP @
    + OVER 3 +
    DO
      3 PICK 1 @ =
      IF
        DUP 1+ WDNM LEAVE
      THEN
      2
    +LOOP
    2-
  REPEAT
THEN
;
```

Dear JAUC,

Having seen Mr. Lechocki's letter in the last Forth User, I felt that I should write in to reassure anyone who is thinking of buying the Alphacom 32 printer. It works fine and should be preferable to the Sinclair since it is now the official ZX printer. Anyone who is interested will be pleased to know that a separate power supply is included, so you do not need to upgrade the Ace power supply, though you will need to add an extra plug.

I also include a couple of Forth words below that may be of interest to club members. A brief description is given underneath the listings.

```
: ULIST
15441
BEGIN
  CR 11 0
  DO
    CR 1-
    BEGIN
      1+ DUP C@ DUP 127
      AND EMIT 128 AND
    UNTIL
    1+ DUP 5 + @
    ."   Defined by  "
    2- DUP @ +
    BEGIN
      1+ DUP C@ DUP 127
      AND EMIT 128 AND
    UNTIL
    DROP DUP @ + DUP
    HERE 1- > 15417 @
    IF
      OVER 15417 @ 1- >
      OR
    THEN
    IF
      LEAVE
    THEN
  LOOP
  DUP HERE 1- > 15417 @
  IF
    OVER 15417 @ 1- > OR
  THEN
  IF
    1
  ELSE
    RETYPE 32 WORD 1+ C@
```

```
   95 AND ASCII N =
  THEN
UNTIL
DROP CR
;
```

This word can be used instead of VLIST. It lists out the user vocabulary in blocks of 11 with the defining word written out afterwards. If you wish to stop the program, just enter N or n when the Ace requests an input. Please note that the program will not work with vocabularies.

```
: M
HERE CONTEXT - 30 -
U.
;
```

This just says how much memory has been used in the Ace.

```
: FR
15384 @ HERE - 1024
/MOD ABS . 15388 DUP
@ 1- SWAP ! ." K"
ABS .
;
```

This just tells the user how much memory is left in Kbytes. Eg. 22K523

Finally I would just like to say that if anyone out there can employ a 19 year Forth enthusiast I would be pleased to hear from them.

Yours faithfully,

A.F.Stratton,
Harcourt fm,
Zeals,
Warminster,
WILTS.

Note from E.H.- This letter as it came to us was printed on an Alphacom printer. The print is generally clear, although very faint for some characters, and with rather variable letter boldness. It uses thermal paper, giving (in this case) blue letters on shiny white paper, and is certainly reasonable value for the money. If you can afford something better, of course.....

25 Friars Walk,
Newcastle,
Staffs,
ST5 2mA

Dear John,

In reply to two letters in the last FORTH USER, I can confirm that the ALPHACOM 32 Printer can be used with the ACE and TAPE 11, as shown by the printing of this letter.

The problems that Mr Kennedy has had could be due to his printer and not his ACE. I had problems with floating point functions when the printer was connected. These were traced to the D6 line from the printer, which is used to indicate that the printer exists, and by putting a diode in line with this (the green wire in the Alphacom) the problems stopped.

I have made use of the D6 line to remind me to turn the printer on by modifying P1 on TAPE 11 as follows :-

```
: P1
 251 IN 64 AND 64
 =
 IF
  22 0 AT ." Printer off "
  ABORT
 ELSE
  MCPRINT 15401 !
 THEN
;
```

Yours sincerely,

(Richard Yorke)

E.M.E. SOUND BOARD REVIEW

Hardware :

The board comes in a sturdy plastic case, about 120 x 100 x 45 mm (LxWxH), with a 50mm cable out of the front for connection to the ACE, a slot to the left for the I/O Ports, and another slot at the back with a duplicate of the ACE expansion bus. On top there are a speaker grill, a volume knob, a reset button (for the sound board only) and a 3.5 mm jack socket to connect another speaker, or to go to an amplifier. The quality of construction is very good and solid, and I can see no problems occurring with wobbly connectors. The only (minor) problem is that it extends the ACE backwards by some 17 cm, which may cause problems if your table isn't wide enough.

Software :

A 24 page manual is provided, giving specifications, details of the chip, including some I/O circuits, and a lot of small words to make the chip easier to use. The instructions are very clear, and all the words I was able to try worked first time. A word of warning - if you type in all these words, which I would recommend, then you won't have enough room in 3K ram to do anything else. A rampack is needed, if you don't have one already (these are also available from E.M.E.). Words are provided to turn on and off channels (there are three tone channels and one noise - see ACE User 4, pages 8-10 for further details as the same chip is used), to program envelopes and to calculate the numbers required for a specified pitch, plus some demonstration sounds (train, trimphone etc.).

uses Sound CHIP AY3-8910 (GI)

Conclusion :

This is a very well produced product, and definitely recommended most highly.

(E.H.)

(now available from Boldfield Computing @£39.10 + carriage)

Book Reviews - by E. Hattersley

'Exploring Forth' by Owen Bishop £6.95 from Granada Publishing

This is a nicely produced book, and quite cheap, but I found it rather verbose, with rather long-winded explanations of eg. the stack. It is written assuming that you have a BBC or an Electron, which would be confusing for someone with a different micro. Its best feature is the large number of practical examples (a simple game, sorting routines etc.), which are used to explain Forth words, and which give some ideas for experimenters. If you have a BBC or Electron then this would be a good book to type in examples from, but otherwise it makes somewhat confusing and tedious reading. Readers with other micros would do better with Steve Oakey's book.

'Forth for Micros' by Steve Oakey from Newnes Technical Books

This is a nice book, written for people with a knowledge of Pascal or Basic, although people with only micro experience may find some of chapter one confusing, with its references to compilers and interpreters. Once over this, however, it is a concise book, with a lot of good material on arrays and data types. It suffers, as usual, from standardisation problems, but there is a chapter on different versions of Forth for the ACE, Dragon and FIG-Forth. The book itself uses only Forth-79. If you are learning Forth as a second language then this book is recommended, and is more readable than Bishop's.

'Advanced Software in Robotics' from Elsevier Science Publishers

Well, I was asked to review this book, so I suppose I'd better, but frankly I didn't understand most of it, and I doubt if anyone not doing research into fifth generation robots would either. However, if you are doing research into fifth generation robots then it is probably a very worthwhile book to have. (In case anyone reading this is, the book is a series of papers, presented at an International Meeting held in Liege, Belgium, on various aspects of robotics - modelling, control, languages etc.)

'Mathematical Elements for Computer Graphics' by D.F.Rogers and J.A.Adams £18.25 from McGraw-Hill

If you got this book, as I did, expecting a series of recipes for fast, 3-D graphics, then you'd get a shock. The book is basically a maths textbook, giving a complete mathematical derivation of some graphics techniques. Nothing here on how to get high-resolution graphics on your ACE, but plenty on, for example, curve fitting, plane fitting (to a series of specified points), fast (matrix) methods for scaling, rotation and moving of objects in 2 and 3-D, and for projecting objects. There are some algorithms in the back (written in a rather obscure version of Basic : Dartmouth Sixth Edition), but basically this is a book for you to work out your own programs, given the mathematical techniques presented. The maths is fairly easy (mainly matrices) as the course the book is intended to cover is a first-year university course. This book is highly recommended for anyone seriously interested in good computer graphics.

(I am working on some ACE words to implement some of the routines in the book, which may be published in Forth User in the future.)

Reading and Writing a Raw Bytes File (Expanded Ace)

Mike Greenwood

I wanted to read any part of memory and to write and edit in the reserved Ram above Ramtop the lengthy bytes file my project requires, and I wrote this suite to do it. I also find it useful for inspecting and fiddling with the spelling of user-defined dictionary words and the operating system as it can also be used to change bytes in Ram below Ramtop.

RAW (Read And Write) displays on screen a line of chars. from memory each side of the imaginary pointer, with an uparrow on the next screen line to indicate the current pointer position. The rest of the screen displays information about values at the pointer and the endpointer which marks the end of the file.

When RAW is executed, in addition to this display,until exitted by <break> , it responds to most keypresses by writing that keyboard char. into Ram at the pointer which moves on one byte, displaying the result and recurring so that it is possible to type continuous text into Ram. If you try to do this below Ramtop your typing is ignored, so you have to start byreserving some Ram with eg 20000 DUP RTP ! PTR ! which also puts the pointer there. Some keypresses are reserved for controls;-
Shift 5 moves the pointer back one and shift 8 moves it on.Shift 6 & 7 move it on or back 100 bytes for coarse adjustment of the pointer position, but of course you can also use "(n) PTR +! " .
Shift 0 will delete the byte at the pointer (but not if the pointer is below Ramtop)and shift the whole of the remainder of the file up to the endpointer back one byte, and shift 1 will wipe the screen which is handy for getting rid of screen clutter caused by encountering 13 bytevalues. Speed of execution is such that the recursive action of RAW allows repeated action without unwanted entries and deletions.Moving up and down memory with shifts 5 and 8 has the chars. from memory chugging across the screen like a train, which is pretty to look at and also pretty useful as it allowscontinuous reading from memory.The endpointer does its best to keep track of where the end of your bytes file is meant to be but you do need to watch it and if necessary adjust with "(n) EPR +! ".

Having exitted from RAW with <break> a bufferful of text up to any chosen delimiter can be inserted into Ram (provided you are above Ramtop) immediately after the pointer. The syntax for this is, eg, "ASCII & INSERT textforinsertion&". The pointer is left at the end of the insert, ready for the next; the result is displayed, the remainder of the file up to the endpointer is shifted up in Ram to make room for the insert and the endpointer adjusted. My inserts don't contain #, so I use an easier development of this for insertions -

" INS# text for insertion which can include spaces# "

Single bytes can be written into Ram at the pointer even if it's below Ramtop with "(bytevalue) WRIB", and the result is displayed (if your tinkering has'nt crashed the system), so you can write in the bytes used as control chars. by RAW. If you want to INSERT a single byte (only above Ramtop) it's best to insert anything eg "INS# X#, "use RAW to move the pointer back and "(bytevalue) WRIB" the byte into place.

Listing

```
: R REDEFINE ; ( We all have this next to FORTH, don't we ?)
CMOVE ( you need a proper version like G.R.YORKE'S in A.U. no: 4 )
+! ( as per manual p.44 )
Ø VARIABLE PTR ( the same within-word temporary pointer  )
Ø VARIABLE EPR  ( endpointer marking EOF )
(If some of these were'nt already in my toolbox I might not
define them separately -when does it become worthwhile?)
9216 CONSTANT FSA ( First Screen Address )
9952 CONSTANT LSA ( Last Screen address/cursor )
15384 CONSTANT RTP (        address of store of RAMTOP )
: H 16 BASE c!   ; ( some say "HEX" but I prefer tiny names for- )
(-oft-used toolbox words to save on the typing-who's right ?)
: B 2 BASE C!   ; (BINARY  )
: C INVIS CLS     ; ( screenwipe )
: .Ø  H  ." Ø "  .  DECIMAL  ; ( prints TOS.as a Hex no; )
:  P  PTR  @     ;   ( puts pointer position onto TOS.)
:   O   -1   PTR +!  ; ( moves pointer back one )
:   Q   1    PTR   +! ; (moves pointer on one )
:   W   25Ø  Ø  DO LOOP ; ( a short wait )
```

```
: WINK   ( winks two chars. against each other at the address )
 DUP  ROT   SWAP  3  1  DO C!  W   LOOP  ;
: K  ( indicates word being executed is waiting for a keypress)
BEGIN   32  75 LSA    WINK  INKEY   ?DUP   UNTIL    ;
:   EPR+   ( updates the endpointer   )
EPR    @   P   1+ MAX EPR  !          ;
:   PD1   ( first part of pointer info display   )
."  RAMTOP =  "   RTP   @  .  CR  CR
."  POIN.TER=£££££££££££££ENDPOINTER=" ( £= space )
CR   EPR   @  P   OVER   OVER   .   14   SPACES   .   CR
.Ø   14 SPACES  ' .Ø   CR   CR   CR
."  C@£££££££££££@ "   CR    P   @   P   C@  OVER OVER OVER OVER
DUP   EMIT   CR   .   8  SPACES   .   CR
.Ø   8   SPACES    .Ø    CR
B   .    4  SPACES    .   DECIMAL  ;
:  PD2    ( second part of pointer info display  )
18  ØØ     AT    P   14   -     29   .  TYPE
19   14     AT  94    EMIT    CR    ;


:   P?   ( pointer info disp lay )
Ø   Ø    AT     PD1     PD2      ;
:  WRIB   ( write byte on T.O.S. into Ram at the pointer )
P     C!     EPR+    Q     P?   ;
:   RAW    ;   ( a dummy to allow PRAM definition  )
:   PRAM    ( protects Ram below Ramtop  )
RTP    @   1+  ABS    P  >  IF    RAW   THEN    ;
:  DELB   ( deletes the byte at the pointer,   shifts rest of file)
PRAM   P   Q   P   SWAP   EPR   @   P   -
CMOVE   -1   EPR   +!  0    RAW     ;
:  RAW    ( main word - Read and Write into memory   )
p?   K
DUP    1   =   IF   0   RAW   THEN
DUP   7   =   IF  -1ØØ   PTR   +!   RAW   THEN
DUP   9   =   IF  1ØØ   PTR   +!  RAW    THEN
DUP   5   =   IF   DELB     THEN    ( How much  time & memory- )
DUP    3   =   IF   Q   RAW    THEN  (- does the use of CASE - )
DUP   1Ø    =   IF   C   RAW   THEN  ( - actually save in this -)
PRAM   WRIB    RAW  ;          ( - sort of situation ? )
R  RAW    ( get rid of dummy RAW  )
```

→

Corrections to the SCRN*/SCR suite (vol 2 no:1 pp 12-13)

(p 12 line 2)-(leave time to quit (enter)) (insert-)
9216 pr ! 79 CU ! (this puts the pointer on the screen-)
(-and a visible character in the cursor, which helps !)
(p 12 line 13-) DUP 11 < if DUP (- delete the final DUP which-)
(- is us. and only leaves a pile of junk on the stack)
(p 13 line 16-) (should read -)
42 995? c! (mark end of screen)

I now want to rewrite this suite in terms of more portable smaller words with as much standardisation with other users as possible. It is both more economical in the long run and makes listings more readable to use eg

9216 CONSTANT FSA (First Screen Address)
9952 CONSTANT LSA (Last Screen Address), and the routine to put the pointer back on the screen if it has wandered off is just one special case of the gene ral problem of resetting a variable to top or bottom limit of a defined range if it has escaped from the range limits; we need a word INRANGE where " variablename LIMIT1 limit2 INRANGE " will reset the variable to limit1 (whether its the max or the min) if it is outside the range.

(MIKE GREENWOOD)

(from 30)

```
: INSERT   (eg ASCII   £ INSERT  Hello Everybody !£   )
PRAM   WORD   C@   Q   P   DUP     3   PICK     +   EPR   @
P   -   CMOVE   (   make room for insert )
PAD   1+   P   3   PICK   CMOVE   DUP   PTR   +!
EPR   +!   0   p?   ; (move insert from PAD to pointer)

: INS##  ( eg  INS## Evening All## )
ASCII  ##  INSERT   ;
```

Mike Greenwood.

(P.S Any BASIC enthusiast like to emulate this word -in less than 4k and six months ?)

D. KEATES
35, THE WALK,
FELIXSTOWE,
SUFFOLK.

DEAR SIR,

I HAVE JUST READ A MATES COPY OF YOUR MAG. AND LIKE IT.

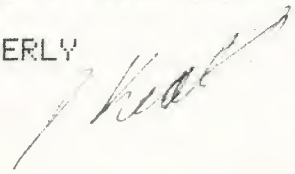
R. HILTON,S PROG. IN SPRING 83 EDITION PROMPTED ME INTO WRITING A MORE USEFUL PROG. WHICH SCROLLS A MESSAGE CONTINUALLY ACROSS THE SCREEN. THE MESSAGE CAN HOLD UPTO 254 CHRS. PUNCTUATION CAN ALSO BE USED, BUT THE INVERSE AND GRAPHIC CHRS CANNOT.

THE WORDS, CODE AND DATA SHOULD BE THE FIRST WORDS IN MEMORY AND IF YOU ARE ONLY USING THE BASIC MACHINE, THE HEX CODE SHOULD BE ENTERED AFTER THESE WORDS, SO YOU STILL HAVE ROOM FOR A HEX LOADER TO ENTER IT WITH.

IF YOU DECIDE TO PUBLISH THIS PROGRAM IT MUST BE WORTH A YEARS FREE SUBSCRIPTION OF YOUR MAG. IF NOT THEN USE IT ANYHOW.

GOOD LUCK WITH YOUR MAGAZINE.

YOURS SINCERLY



```
CREATE CODE 70 ALLOT  OK
CREATE DATA 0 , 0 , 0 , 0 ,  OK
: MOVE
 CODE 52 + CALL
;
  OK

: PAUSE
 500 0
 DO
 LOOP
;
   OK

: SCROLL
 4 0
 DO
  CODE CALL PAUSE
 LOOP
MOVE
;
   OK

: COPYUP
 DATA SWAP DUP 8 +
 SWAP
 DO
  I C@ OVER C! 1+
 LOOP
 DROP DUP 95 < SWAP
 62 > AND
 IF
  0 DATA 7 + C!
 THEN
 0 DATA C!
;
   OK
```

```
: FINDC
 DUP 1+ C@ DUP 32
 - 7 * 7546 +
 OVER 94 >
 IF
  32 -
 ELSE
  OVER DUP 62 >
  IF
   63 - -
  ELSE
   DROP
  THEN
 THEN
 COPYUP
;
  OK


: MES
 CLS ." TYPE IN YOUR MESSAGE END
ING IT "
 CR ." WITH THE SYMBOL ~ ( SHIFT
ED A ) "
 CR ." THEN PRESS THE ENTER KEY.
"
 QUERY ASCII~ WORD
;
  OK


: BORDER
 32 0
 DO
  ." *"
 LOOP
;
  OK
```

```
: RUN
 MES CLS ."   LARGE CHARACTERS B
Y D.KEATES "
 BORDER 7 0 AT BORDER
 128 SPACES BORDER
 BEGIN
  FIND<C SCROLL 0
 UNTIL
;
   OK
```

THE WORD CODE = 15452 AND SHOULD BE THE FIRST WORD IN MEMORY.

THE FOLLOWING HEX CODE SHOULD BE LOADED FROM 15452 DECIMAL OR 3C5C HEX.

```
3C5C 11 00 25 21 01 25 01 80
3C64 00 ED B0 11 1F 25 21 BD
3C6C 3C 06 05 AF CB 16 17 CB
3C74 16 17 23 CB 16 17 CB 16
3C7C 17 23 E5 21 A2 3C 85 6F
3C84 ED A0 21 1F 00 19 EB E1
3C8C 10 E1 FD E9 21 01 27 11
3C94 02 27 4E 06 00 1A 23 23
3C9C 0B ED B0 12 FD E9
```

HEX LOADER P. 14 ACE USER SPRING 83.

D.KEATES
35,THE WALK,
FELIXSTOWE,
SUFFOLK.

DEAR SIR,

A FEW MORE LINES FROM ME TO LET YOU KNOW THAT I MADE THE SOUND GENERATOR LEX VAN SONDEREN DESIGNED (WINTER 83 PAGE 6)AND IT WORKED WITH NO PROBLEMS. I DID HOWEVER, CONNECT THE JOYSTICK UP DIFFERENTLY AS THE AY-3-8910 HAS INTERNAL PULLUP. I LEFT OUT THE RESISTORS AND CONNECTED THROUGH THE SWITCHES TO 0V. THE RESULT IS THEN 255 - THE VALUE OF SWITCHES WHICH ARE ON.
I ENCLOSE A LISTING OF A PROGRAM TO USE WITH IT. IT WILL JUST FIT INTO THE 3K MACHINE WITH NO ROOM FOR COMMENTS. THE DISPLAY AND THE INSTRUCTIONS ARE SHOWN BELOW.

YOURS SINCERLY

D Keates

KEY S CHANGES THE SPEED
KEY 1 MOVES THE ARROW
KEY 3 TRIGGERS THE ENV. SHAPE
KEY 6 INCREASES CONTENTS
KEY 7 DECREASES CONTENTS
KEY 0 RESETS ALL REGISTERS

THE BITS IN REG 7 ARE DISPLAYED AT THE BOTTOM OF THE SCREEN.

SG RUNS THE PROGRAM.

DISPLAY :-

REG	CONTENTS	USE
0	0	A TONE FINE
1	0	COARSE
2	0	B TONE FINE
3	0	COARSE
4	0	C TONE FINE
5	0	COARSE
6	31	NOISE PERIOD
7	7	ENABLE LOW
8	16	A VOLUME
9	16	B VOLUME
10	16	C VOLUME
11	0	ENV. PERIOD FINE
12	80	COARSE
13	0	ENV. SHAPE
>14	255	I/O PORT A
15	255	I/O PORT B

IB	IA	NC	NB	NA	TC	TB	TA
0	0	0	0	0	1	1	1

N. B.
IF THE ARROW IS POINTED AT AN I/O PORT IT WILL GIVE A CONSTANT UPDATE OF THE CONTENTS.

SOUND GENERATOR TOOLKIT

- D. KEATES

```
0 VARIABLE R  OK

0 VARIABLE W  OK

: SET
 SWAP 221 OUT 223 OUT
;
  OK

: R?
 221 OUT 221 IN
;
  OK

: R.
 2 0 AT 16 0
 DO
  I 0 SET SPACE I
  DUP DUP . 2+ 8
  AT R? . 2 SPACES
  CR
 LOOP
;
  OK

: RR
 R @ DUP
;
  OK


: K67
 RR R? ROT + SET
;
  OK

: K1
 RR 2+ 0 AT SPACE
 1+ 15 AND R !
;
  OK
```

SOUND GENERATOR TOOLKIT

- D. KEATES

```
0 VARIABLE R  OK

0 VARIABLE W  OK

: SET
 SWAP 221 OUT 223 OUT
;
  OK

: R?
 221 OUT 221 IN
;
  OK

: R.
 2 0 AT 16 0
 DO
  I 0 SET SPACE I
  DUP DUP . 2+ 8
  AT R? . 2 SPACES
  CR
 LOOP
;
  OK

: RR
 R @ DUP
;
  OK


: K67
 RR R? ROT + SET
;
  OK

: K1
 RR 2+ 0 AT SPACE
 1+ 15 AND R !
;
  OK
```

```
: SP
 800 W @ - W
 !
 BEGIN
  INKEY 0=
 UNTIL
;
  OK

: T
 16 15388 @ - 31
 AND SPACES
;
  OK

: A
 ."  TONE FINE"
 T ."   COARSE"
 T
;
  OK
: SETUP
 CLS ." REG    CONTENTS    USE"
 CR CR T ." A"
 A ." B"
 A ." C"
 A ." NOISE PERIOD"
 T ." ENABLE LOW"
 T ." A VOLUME"
 T ." B VOLUME"
 T ." C VOLUME"
 T ." ENV.PERIOD FINE"
 T ." COARSE"
 T ." ENV.SHAPE"
 T ." I/O PORT A"
 T ." I/O PORT B"
 R.
;
  OK
```

```
: SG1
 INKEY DUP 49 =
 IF
  K1
 ELSE
  DUP 55 =
  IF
   -1 K67
  ELSE
   DUP 54 =
   IF
    1 K67
   ELSE
    DUP 48 =
    IF
     R.
    ELSE
     DUP 115 =
     IF
      SP
     ELSE
      DUP 51 =
      IF
       13 13 R? SET
      THEN
     THEN
    THEN
   THEN
  THEN
 THEN
 DROP RR 2+ DUP 0
 AT ." >"
 OVER . 8 AT R?
 . 2 SPACES
;
  OK

: SG
 SETUP CR ." IB IA NC NB NA TC T
B TA"
 BEGIN
  FAST SG1 7 R? 24
```

```
  0
  DO
   21 22 I - AT
   2 /MOD SWAP . 3
  +LOOP
  DROP SLOW W @ 0
  DO
  LOOP
  0
 UNTIL
;
  OK
```

I made a word PLIST, to display the names of the words defined in addition to the standard dictionary (thus a VLIST except the standard words). It is useful in editing great vocabularies:

```
: PLIST CONTEXT @ @
 BEGIN 2- DUP 2+ DUP C@ 4 + -
       BEGIN 1+ DUP 1- C@ DUP
             127 AND EMIT 127 >
       UNTIL
       SPACE DROP @ DUP 9000 <
 UNTIL DROP ;
```

Perhaps it is something for a next FORTH USER.

Sincerely yours,

Herman Hillebrand
(Netherland)

10 Sharpthorne Crescent
Portslade
BRIGHTON.
10 July 84

Dear John,

a quick word of advice for anyone with a PACER ram pack. I'd been having trouble with my ACE crashing because the internal regulator was getting too hot. I hooked up an external regulated 5V supply and tried to use the existing power pack for the RAM (remember to cut the connections to the regulator if you try this). It was soon apparent that the D4 line was sometimes reading Ø when it should have been 1, causing all manner of spurious output, etc.

Having read my past issues of ACE User I reduced the unregulated 9v with a regulated 12V. The errors became more frequent! In desperation I reduced the regulated supply to 9V. Bliss!

Moral: The PACER uses 5290 chips, not 4116s, and it doesn't like 12V.

Yours sincerely,

Ken Moffat (Ken Moffat).

P.S. I've got the same problem as John Kennedy (Vol 2 No.1 p20). I prefer small capitals to the condensed lower case characters, so I've got a bytes file for 'a' to 'z'. About one third of the times I load it I find upper case corruption or corruption of 'Ø'. Anybody know the answer?

ON-SCREEN INPUT (ACE) - Colin Dooley

Have you ever wanted an on-screen input for your Ace, without all that mucking about with NUMBER, WORD, RETYPE etc.
Here is my solution:

```
0 VARIABLE CHARS          (No. of characters entered)
0 VARIABLE INPUTLENGTH    (No. of characters allowed)
0 VARIABLE FIRSTCHAR      (range of characters that will
0 VARIABLE LASTCHAR        be accepted)
15338 CONSTANT SCRPOS     (system variables)
15339 CONSTANT KEYCNT

: KEYWAIT                 (Wait for key to be released or to
1000  0                    repeat)
DO
LOOP
BEGIN
 INKEY  0= KEYCNT C@ 4
 =
 IF
  5 KEYCNT C! 1
 ELSE
  0
 THEN
 OR
UNTIL
;

: .CRSR      (Draw a cursor (underline character) on the
              screen)
  ASCII _ EMIT SCRPOS @ 1-
  SCRPOS !
;
```

```
: DELETE          (Delete last character entered if
                   CHARS > Ø)
 CHARS @ Ø>
 IF
  CHARS @ 1- CHARS !
  SPACE SCRPOS @ 2- SCRPOS
  ! .CRSR
 THEN
;


: DELETE?         (Delete or delete line pressed?)
  INKEY DUP 5 =    (Delete)
  IF
   DELETE KEYWAIT                                    ;
  THEN
  10 =                    (Delete line)              :
  IF
   CHARS @ Ø
   DO
    DELETE
   LOOP
   KEYWAIT
  THEN
;
: INPUT           (Do the actual input)
                  (Ends with the address of the start)  ;
                  (of the data on screen on the)
                  (stack.)                              :
Ø CHARS ! .CRSR KEYWAIT
BEGIN
 INKEY DUP 13 =    (Enter pressed)
 IF
  DROP SPACE SCRPOS @ CHARS
  @ - 1- EXIT
 THEN                                                  ;
 DELETE?
 DUP 11 >
 IF
  DUP DUP FIRSCHAR @ -     (Check if character is
  -1 > SWAP LASTCHAR @      (in legal range.)
```

```
    - 1 < AND CHARS
    @ INPUTLENGTH @ < AND
    IF
      EMIT .CRSR CHARS @ 1+
      CHARS !
    ELSE
      300 100 BEEP DROP          (IF not, beep )
    THEN
    KEYWAIT
   ELSE
    DROP
   THEN
   0
  UNTIL
;

: INPUTSTRING
  250 INPUTLENGTH !          (Accept up to 250 characters.)
  32 FIRSTCHAR !             (In range 32 to 127)
  127 LASTCHAR !
  INPUT CHARS @ 0
  DO                         (Move string to PAD )
   I OVER + C@
   I PAD + C!
  LOOP
  DROP PAD CHARS @           (leave address, length on stack)
;

: INPUTNUMBER                (Input single length integer)

  5 INPUTLENGTH !            (Accept up to 5 digits)
  ASCII 0 FIRSTCHAR !        (In range 0 to 9 )
  ASCII 9 LASTCHAR !
  0 0 INPUT 1- CONVERT
  DROP DROP                  (leave number on stack)
;
```

The last two words show examples of input and how to use it. The data is inputted on the screen after the last character printed. The keys repeat, and delete and delete line will work.

John kennedy wondered if it was only his ACE which corrupted data as it was written to the character RAM. No, mine does as well, also manages to corrupt areas of this memory wh weren't even the ones being written to! I hav only managed any of these when writing to the memory which waits for the screen. So you can have either corrupted characters or screen glitches John.

Can anybody come up with machine code D* and D/ (Double length versions of * and /).

Colin Dooley

Dear John,

Enclosed you should find my renewal form, and details of some useful routines for the ACE. I hope the routines below are good enough to be reprinted as an article in ACE USER,

Yours sincerely

Stuart Howell.

PS. For those who want speech on the ACE, the easiest way of doing it is to use a Z80-PIO and the General Instruments Speech Synthesizer SP0256-AL2.

See February '84 ETI. (Electronics Today International) for a speech board using this chip. It will need some alteration to fit the Ace's memory map, as the project was originally for the ZX81. The same issue has an article on fitting extra memory to Z80 machines (as in the Ace).

E.H.

Your wish is our command.....

```
: DIR                          ( Lists all directory
CLS 15431 @ 0 15431 !            not in rom, ie. user
VLIST DROP 15431 !                      defined words )
;

DEFINER MCODE                  ( Word to initialise
ALLOT                            M/C routine. Requires
DOES>                            the number of bytes
CALL                             to be used on the
;                                                stack )

: WHERE                        ( Finds the start
( - start address of             address of the code
                code )           in the following word )
FIND 2+
;

: CODELEN                      ( Finds the number of
( - No. of bytes in              bytes in the code in
                code )             the following word )
FIND 5 - @ 7 -
;
```

```
: ?LINE20                       ( Waits for a keypress
15388 @ 9855 >                    when text reaches line
IF                                20 on screen. Used in
  BEGIN                                LOADER and READER )
    22 29 AT ." *"
    INKEY
  UNTIL
  CLS
THEN
;

: INPUT                         ( As ACE manual p96,
( - ? )                           except for replacing
-32768 QUERY INVIS LINE                  QUIT by ABORT )
VIS SWAP -32768 -
IF
  ." Hello, hello, hello, what's "
  CR ." going on here then?"
  ABORT
THEN
;

: TAB                           ( As ACE manual p75.
( tab stop - )                    Used for screen
15388 @ - 31 AND SPACES                  formatting )
;

: READER                        ( Prints a dump of
( address from,                   memory starting from a
  number of bytes - )             given address, for a
OVER + SWAP                       gievn number of bytes,
DO                                stopping after each
  5 TAB I . 15                                screenful )
  TAB I C@ . CR
  ?LINE20
LOOP
;

: LOADER                        ( Allows input of bytes
( address, number                 starting at a given
           of bytes - )           address, for a given
CLS OVER + SWAP                   number of bytes. Start
DO                                address is usually
  5 TAB INPUT DUP .                       given by WHERE )
  I C! 12 TAB I .
  ?LINE20
LOOP
DECIMAL
;
```

```
: HEX                          ( Changes system number
16 BASE C!                       base to hexadecimal )
;
```

Machine code routines
------- ---- --------

These are defined by:-

n MCODE name where: n = number of bytes

name = name of routine

and then using

WHERE name n HEX LOADER

```
SCREENSAVE ( 11 bytes )        ( Saves the screen to
                                 a block of memory
RST 24          DF               given by the address
LD HL,8192      21 00 20         on the stack. This
LD BC,768       01 00 03         can be put on the
LDIR            ED B0            stack by using a
JP (IY)         FD E9            word such as TVPIC,
                                 which has been
                                 defined by using

                                 CREATE TVPIC 768
                                 ALLOT                )

RECALL ( 12 bytes )            ( Restores a stored
                                 screen to the screen.
RST 24          DF               Picture is recalled
EX DE,HL        EB               from the address on
LD DE,8192      11 00 20                    the stack )
LD BC,768       01 00 03
LDIR            ED B0
JP (IY)         FD E9


MOVE ( 12 bytes )              ( Copies a block of
                                 memory:
RST 24          DF
PUSH DE         D5               T.O.S.  No. of bytes
RST 24          DF               o f t   Where to
PUSH DE         D5               p   a   Where from
RST 24          DF                   c
EX DE,HL        EB                   k   This is useful
POP DE          D1                       for graphics
POP BC          C1                       memory moves.
LDIR            ED B0                    Be careful if
JP (IY)         FD E9                    moving over-
                                         lapping areas )
```

```
ALTER ( 21 bytes )                    ( Used in CHANGE
                                          and INVERT )
          RST 24          DF
          LD HL,9216      21 00 24
          LD BC,768       01 00 03
:L2       LD A,E          7B
          CPIR            ED B1
          JR NZ,:L1       20 07
          DEC HL          2B
          LD (HL),D       72
          INC HL          23
          LD A,B          78
          OR C            B1
          JR NZ,:L2       20 F4
:L1       JP (IY)         FD E9


: CHANGE                ( Changes all occurences
( n1,n2 - )               of character n1 on
256 * + ALTER                     screen to n2 )
;

: INVERT                ( Inverts all occurences
( n1 - )                        of n1 on screen )
DUP 128 +
256 MOD
CHANGE
;
```

SAVE and LOAD of single words

(by Morten Levy)

As an ACE user I often meet the problem of handling words independently to the dictionary: you have a program of, say, twenty words, six of which are solutions to very general problems. How can you pick those six words from that dictionary to load them into another? That is very difficult with the ACE (and for that reason I think it was a mistake of the ACE manufacturers not to maintain the FORTH concept of Screens).

(Note from E.H.- Good point, but it just won't work without a disk-drive, which costs a lot.)

With two limitations the routine that follows shouls solve the problem. The two limitations are:
1) The words you save of load should not be larger than one screenful when LISTed. In a way that is an advantage as it encourages you to keep your words small.
2) When you use this routine to load some words into a dictionary you will find the words of the routine in the dictionary too. That can be annoying. It is possible to avoid it, but it is not simple. (More on this later.)

The words of the routine are STORE and FETCH. Both words use W-INPUT which accepts a string of words you want to save of loas and puts it in safety in PAD1. This string should consist of word-names and spaces alternately. Be careful that you leave one space and one space only between the word-names, as the routine takes two neighbouring spaces as a sign that you have finished you input. Also, the inverse copyright (ascii 255) should not be used, as this is used as a dummy delimiter for WORD. The routine can handle a maximum of 253 input characters.
The word INITIALIZE used by both STORE and FETCH sets the pointer POS to the start of PAD1 and put spaces in the end of PAD1 to ensure smooth functioning.

On STORE: The routine will input the list of word names (W-INPUT), and then repeatedly execute ST-PROCESS until PAD1 is empty. ST-PROCESS takes a word-name from PAD1 and puts it into the input buffer (BUF), LISTs the word (STORE1), moves the same word-name once more to the input buffer (BUF), BSAVEs the relevant part of the screen under the word-name in question (STORE2), and, ultimately (back in STORE) does the same with the next word-name, and so on.

On FETCH: The routine will input the list of word names (W-INPUT), and then repeatedly execute FE-PROCESS until PAD1 is empty. FE-PROCESS moves a word from PAD1 to the input buffer (BUF), BLOADs the word to the screen (FETCH1), accepts the listing as input by changing the value stored in address 15396 (ie. the start of the input buffer) (FETCH2), and then (back in FETCH) does the same to the next word-name, and so on.

Error-handling: If you try to save a word not contained in the dictionary the routine will stop and ERROR 13 will appear. You'll have to enter VIS as you are in INVIS mode.
If you try to load a word which contains a word not found in the dictionary the routine will stop and wait for you to correct the word, as in a normal EDIT session. You can try to correct it and then continue, but often the easiest way out is to enter [ABORT] and then VIS, as you are in INVIS mode.
If you have tried to save a word that is too large for one screenful and then later attempt to load it, you'll just get a word-torso on the screen and the ACF will probably crash. So don't do that.

Clean bill: When you fetch some words you'll have the problem mentioned above - how are you going to get rid of the words of this routine? The best thing to do, naturally, is to rewrite the routine into machine-cod above RAMTOP (in fact, I have done that - ask John Noyce is you are interested). There are two other possibilites:

1) Loading by hand (for a few words)

Enter

```
FORGET <first word>
```

then, for each word you want to load enter

```
INVIS CLS 9281 0 BLOAD <word-name>
```

start tape, then enter (when found)

```
32 9280 C! 9280 15396 ! LINE VIS
```

2) Loading via REDEFINE (several words)

Enter

```
FORGET <first word>
```

LOAD a dictionary consisting of two words:

```
: R REDEFINE ;
: X 58 9960 C! 32 9961 C! 121 9962 C!
   32 9963 C! 59 9964 C! LINE ;
```

in that order. Then

```
X <enter> X <enter> .....
```

as many times as the number of words you are going to load minus 2. Load the routines given here, then load the words required via FETCH. Enter

```
R X R X .....
```

as many times as the number of words you have loaded minus 2. Enter

```
R X R R FORGET <first word of routines given>
```

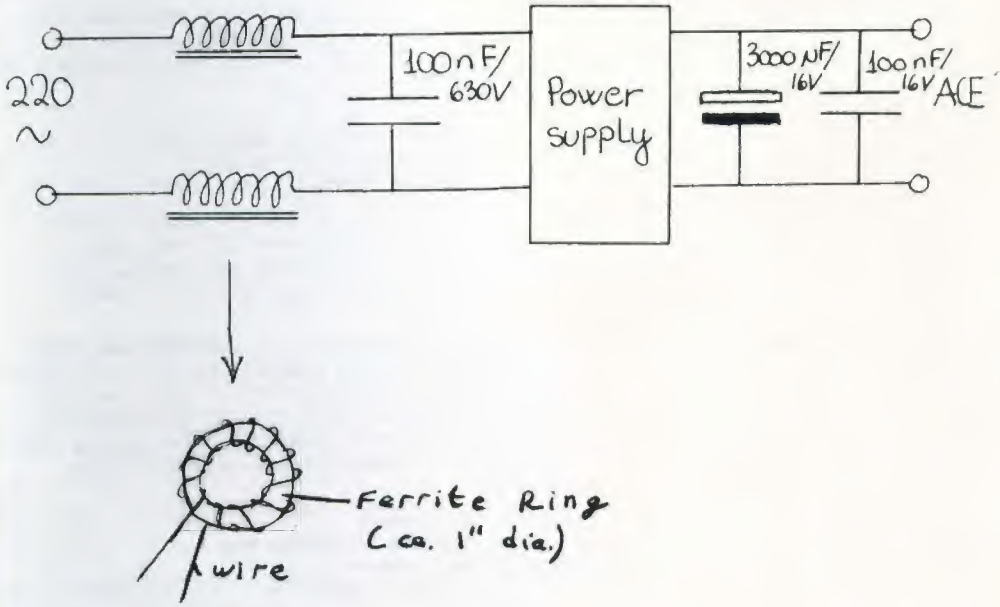
Now your dictionary should consist of the desired words only.

```
0 VARIABLE PAD1 253 ALLOT
0 VARIABLE POS

: INC
 DUP @ 1+ SWAP !
;

: CMOVE
 ( adr1 adr2 n -)
 DUP 1 (
 IF
  DROP DROP DROP EXIT
 THEN
 0
 DO
  OVER I + C@ OVER
  I + C!
 LOOP
 DROP DROP
;

: INITIALIZE
 PAD1 DUP POS ! 253
 + 0 !
;

: W-INPUT
 QUERY 255 WORD 1+ PAD1
 253 CMOVE
;

: W-COND
 ( -flag)
 POS @ C@ 32 =
 0=
;

: BUF
 9957
 BEGIN
  W-COND
```

```
 WHILE
  POS @ C@ OVER C!
  POS INC 1+
 REPEAT
 DROP
;

: FETCH1
 CLS 9281 0 BLOAD LINE
;

: FETCH2
 32 9280 C! 9280 15396
 ! LINE
;

: FE-PROCESS
 CLS BUF INVIS FETCH1 FETCH2
 VIS POS INC
;

: FETCH
 INITIALIZE CLS ." Write the words you want
 CR ." (ENTER)"                          to load"
 CR CR ." Start tape when screen is clear"
 W-INPUT
 BEGIN
  W-COND
 WHILE
  FE-PROCESS
 REPEAT
;

: STORE1
 CLS LIST LINE
;

: STORE2
 8192 15388 @ 9216 -
 BSAVE LINE
;
```

```
: ST-PROCESS
 POS @ CLS BUF  INVIS
 STORE1 POS ! BUF STORE2
 VIS POS INC
;

: STORE
 INITIALIZE CLS ." Write the words you want
 CR ." Start tape"                    to save"
 CR CR ." (ENTER)"
 W-INPUT
 BEGIN
  W-COND
 WHILE
  ST-PROCESS
 REPEAT
;
```

If your Ace still is 'going down' this might help.



Christian Bo Thystrup
Kaervej 12 Hellested
4652 Haarlev
Denmark

Mr. S. Jackson.
51 Wadsley Lane,
Sheffield, S6 4EA.
S. Yorks.

Tel: (0742) 348356

Dear JAUG,

Many thanks for the copy of "FORTH USER vol.2 no.1" which includes my letter on system expansion. As requested, here is an update on the system:-

A problem has been found in the expansion bus boards (used to modify and buffer the Ace bus), which would prevent full DMAs from taking place. The hardware problem has now been overcome and four new boards are being prepared. These boards are as follows:-

I Bus buffers (for normal bus signals)
II Derived bus and control logic (5 commonly used lines)
III Memory paging port (loc FFFFh giving A16' to A19')
IV Page indicator (pretty lights as FFFFh is write only)

An advantage of the use of a number of smaller boards is that users not wanting to implement a particular feature need only omit the unwanted boards. I will send details as soon as the PCBs are working, but, for now, the first of the attached sheets shows my suggestion for a more sensible bus structure.

Secondly, and probably more interesting, is the 16k memory unit the prototype of which is working perfectly on my own Ace. For any interested parties, the working unit is wire-wrapped, although a PCB is being prepared to suit the suggested bus. As soon as this is ready, the foil patterns will be available to club members. For users who want to go thier own way, an SAE will secure the circuit diagram (64k memory map only). However the whole unit provides the following features:-

I Full 16k of dynamic RAM (of course)
II Link/Switch selection of position in memory map
III Link/Switch selection of page number
IV All for less than £15 if you shop carefully

Thirdly, and with reference to quite a few letters in "FORTH USER" here is an idea for those of you who are trying to replace the "soggy sponge" with a real keyboard.

Although Doug Bollen's interface (ETI Nov 83) is very good, there is a simpler way! On the second of the attached sheets are a number of diagrams which show a far cheaper method of doing the same thing INSIDE the case. This should only be attempted by people who are competent with a soldering iron as the guarantee will not survive the process. I use this arrangement myself with a superb Hall-effect keyboard (ex-equip from Chiltern Electronics if they have any left), If you do get one of these, there is a LOT of work to do. I use the Hall I.C.s to control 4066Bs in place of the keyboard matrix. Note: Hall switches do not generate any switch bounce! Any alternative can be used (as in Mr. Bollen's article). A future letter will show how to add extra keys to the Ace keyboard.

Yours Faithfully,

Stephen Jackson 26/1/84

ITEM 1

SUGGESTED BUS STRUCTURE

PSU +12V		1		PSU +12V
PSU +9V		2		PSU +9V
PSU +5V		3		PSU +5V
CLOCK	Φ	4	$\overline{M1}$	INSTRUCTION FETCH
RESET	$\overline{RESET}$	5	$\overline{HALT}$	HALT
WAIT	$\overline{WAIT}$	6	$\overline{RFSH}$	REFRESH
ODD ADDRESS	A1	7	A0	EVEN ADDRESS
	A3	8	A2	
	A5	9	A4	
	A7	10	A6	
	A9	11	A8	
	A11	12	A10	
	A13	13	A12	
	A15	14	A14	
EXTENDED ODD ADDRESS	A17'	15	A16'	EXTENDED EVEN ADDRESS
	A19'	16	A18'	
ACE CONTROL LINE	$\overline{WE}$	17		
MEMORY REQUEST	$\overline{MREQ}$	18	$\overline{RD}$	READ
I/O REQUEST	$\overline{IOREQ}$	19	$\overline{WR}$	WRITE
I/O READ	$\overline{IORD}$	20	$\overline{MEMRD}$	MEMORY READ
I/O WRITE	$\overline{IOWR}$	21	$\overline{MEMWR}$	MEMORY WRITE
BUS AKNOWLEDGE	$\overline{BUSAK}$	22	$\overline{BUSRQ}$	BUS REQUEST
ODD DATA	D1	23	D0	EVEN DATA
	D3	24	D2	
	D5	25	D4	
	D7	26	D6	
		27	$\overline{INTAK}$	INTERUPT AKNOWLEDGE
MASKABLE INTERUPT	$\overline{INT}$	28	$\overline{NMI}$	NON MASKABLE INTERUPT
INTERUPT PRIORITY O/P	INTPOT	29	INTPIN	INTERUPT PRIORITY I/P
PSU 0V		30		PSU 0V
PSU -5V		31		PSU -5V
PSU -12V		32		PSU -12V

The bus uses Ace type edge connectors (but longer). Dual sided 32 ways per side. Note: Maplin supply the elusive mounting brackets for these.

The article by Doug Bollen (ETI Nov 83) describes a peripheral board designed to interface keyboards or joysticks to the Ace. If you do not want to invalidate the guarantee, or you do not feel confident soldering to the Ace PCB then this interface is for you. If on the other hand you are a competent constructor, there is an alternative.

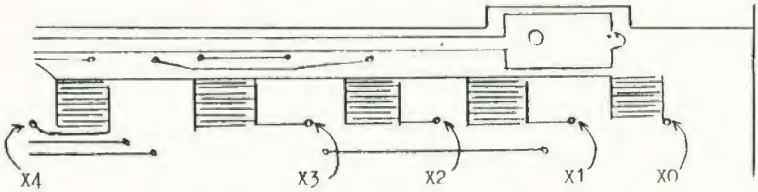
As stated in the article, the interface runs in tandem with the internal keyboard ports. It stands to reason, therefore, that if the relevent locations on the Ace PCB can be found interfacing can be carried out in a far more convenient manner within the machine. This is how to go about it.

1) Dismantle the Ace completely and remove the heatsink.

2) Locate the eight diodes which correspond to the 1N4148s of Mr. Bollen's interface (these are the small yellow components with black bands towards the back of the board which can be found immediately below the speaker).

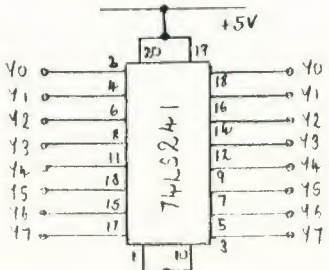
3) Solder fine wires to the keyboard end of each of these diodes. They are Y3,Y4,Y5,Y6,Y7,Y0,Y1 and Y2 in the matrix respectively These wires should be approximately 8" long

4) Look at the PCB where the rubber keyboard is usually sited and use figure 1 to locate points X0 to X4 when you are sure you have found them, solder fine wires into them from the underside and trim the protruding wires flat to the top of the board. These wires should be approximately 8" long.



5) Use double sided tape to hold these wires to the underside of the board and route them through the vacant hole in front of the cassette sockets. Trim the wires to 4" from the board surface.

You are now ready to decide on the details of your new keyboard. If you intend to use a keyboard on a trailing lead, you must build the buffer board shown in figure 2 because the capacitance a long lead will introduce onto the address lines will otherwise stop the system DEAD! You probably wont need to do this if you rebox your ace inside your new keyboard.



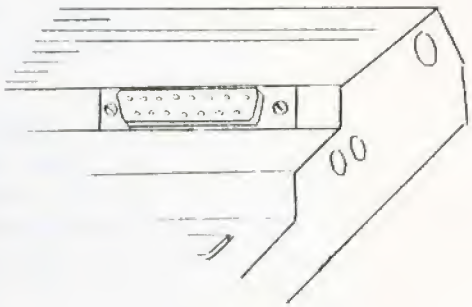
This circuit is best built onto a small piece of Vero-board, the power being extracted direct from the Ace PCB. Power can be taken from the edge connector or from a point close to the site of the buffer within the case. Please note that if you are unfamiliar with the decoupling requirements of TTL buffers, you should include a 100nF capacitor on the buffer board between +5V and Ground, or the operation cannot be guaranteed.

For those people reboxing thier Ace, they must now proceed alone. It is probably better to fit a socket to the Ace and have your keyboard on a trailing lead (you can then attach joysticks, etc. to this). If you decide to do this proceed as follows:-

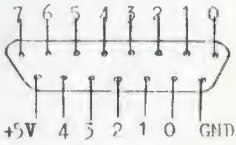
6) Using Blu-tak, stick the buffer board to the top of the group of four TTL I.C.s immediately to the left of the modulator. Route the wires from the diodes against the PCB between the I.C.s. Trim the output wires to a length of 4". Replace the heatsink.

7) Add power supply leads to the Ace PCB if you intend to add any hardware to your keyboard/joysticks.

8) You must now cut a hole in the top of the casing to accept the socket of your chosen connector system. I would recommend a 15 way cannon D-type connector mounted as shown in figure 3.



9) Before mounting the socket, you must feed the wires through the hole in the case. Solder the wires to the pins of your connector I would recommend the following pinout for the 15 way socket.



Y outputs

View from front

X inputs and Power

10) Reassemble the Ace with the connector bolted in place.

At this point you can now test the interface, this is best done by shorting out the socket pins with a 500 Ohm resistor and making sure the character on the screen agrees with the matrix connections detailed in Mr. Bollen's article. If everything is OK you can connect your keyboard to the 15 way socket via a trailing lead. I use a 2m length of 15 way shielded without any problems. (connect the shield at the socket end only).

Finally, MVI will supply copies of the article in the form of back-numbers or photocopies on request.(they cost £1.50 each).

Dear Sirs,

I am thinking of buying a "PSGIO" from Essex Micro Electronics, but in the meantime, if anyone would like to make a joystick interface then just look at page 154 in the manual, and you will see that someone has kindly put in just the circuit you want. I've worked out that it will cost about £ 2 to build, excluding the actual circuit board and joystick.

You can of course make your own joystick quite easily. This is done by taking a square piece of wood (or any material) and drilling a hole about 1 inch diameter in the center. Then cut a circular piece of copper about twice the diameter of the hole and stick it to the wood so that it covers the hole.

Then cut the hole out of the copper again, leaving a circle of copper about ½ an inch wide round the edge of the hole.

Then take a knife and cut away a space leaving 4 quarters of copper around the hole. You then solder to the underside of these copper pieces and take the wires away through the bottom of the wood. Now comes the most expensive part of the whole job. Go down the road and buy a new bolt, (the longer the better).

Get a piece of hardboard (or metal) and cut two pieces to the same area as those of the wooden block. Drill a hole through the middle of both. On what will be the top piece, cut out a hole to a 1 inch diameter. Fix this piece to the top of the wooden block. Take the other piece and put the bolt through the hole you've drilled, then get a small rubber tyre of some sort (from a model or something) and slip this over the bolt. Then get a large washer and slip this on top of the tyre and then get the nut and tighten the nut until the bolt self centres. Solder a wire to the bolt underneath the hardboard and then fix this piece of hardboard to the bottom of the piece of wood. You should now have five wires coming from the joystick. All you need to do now is to go along to your local electronics shop and buy a key to use as a fine button.

This should be connected to the same 5v supply as the bolt and the wire from the key should go with the other 4 switch wires and the one 5v wire. Connect the wires from the copper plates to the 74LS368 along with the wire from the key. Connect the bolt wire and the wire to the key to the +5v supply via a 1K Ω resistor.

Robert Mayell.